Exhibit B

```
// BusinessLayer.cpp : Implementation of CBusinessLayer
#include "stdafx.h"
#include "layerimpl.h"
#include "BusinessLayer.h"
#include "process.h"

class layerthread : public threadFunctor
{
public:
        void startThread()
        {
                COMInitalizer init;
                try
                {
                        handler();
                }
                catch(...)
                {
                }
                m_threadhandle = NULL;
        }
        void handler()
        {
                long threshold;
                long nextChapterTime;
                CTime startEvent;
                CTime stopEvent;
                CTime time;
                long eventLength;
                char msg[256];

                ::OutputDebugString("\n Loop started\n");
                CBusinessLayer *pLayer = reinterpret_cast<CBusinessLayer
*>(m_tParam);
                if (pLayer)
                {
                        try
                        {

                                time = CTime::GetCurrentTime() ;
                                pLayer->get_threshold(&threshold);
                                long timeData;
                                pLayer->get_startEvent(&timeData);
                                startEvent = timeData;
                                pLayer->get_stopEvent(&timeData);
                                stopEvent = timeData;

                                eventLength = CTimeSpan(stopEvent -
startEvent).GetTotalSeconds() * 1000;

                                sprintf(msg, "%s\n", startEvent.Format("%D:%H:%M"));
                                ::OutputDebugString(msg);
                                sprintf(msg, "%s\n", stopEvent.Format("%D:%H:%M"));
                                ::OutputDebugString(msg);
                                pLayer->put_serverTime(time.GetTime(),-1, -1 ,0,
eventLength);

                                if (startEvent < stopEvent &&
                                        (time + CTimeSpan(threshold)) >= startEvent)
                                {
                                        //
                                        // - Determine if it is time to kick-off the
event.
                                        // - If it is, stop the loop.
        // - If it is NOT, go to sleep and check
after 500 milliseconds
        //
        _bstr_t dvdCmd = pLayer->firstdvdCmd();
        std::string debugmsg;
        debugmsg = "first dvd command" + dvdCmd +
"\n";

        ::OutputDebugString(debugmsg.c_str());
        while (time < startEvent )
        {
                checkCancel();
                Sleep(500);
                time = CTime::GetCurrentTime() ;
pLayer->put_serverTime(time.GetTime(),-1, -1 ,0,  eventLength);
        }
        if (time <= (startEvent + CTimeSpan(1)))
        {
                pLayer->sendCommand(dvdCmd);

                ::OutputDebugString("Process
Event");
        }

        while (time < stopEvent)
        {

                BSTR  command;
                CTimeSpan lapsedTime(0);
                long title, chapter;

                if (time > startEvent)
                {
                        lapsedTime = time -
startEvent;
                }

                if
(SUCCEEDED(pLayer->GetNextPair(&nextChapterTime,&title, &chapter, &command)))
                {
                        if
(lapsedTime.GetTotalSeconds() < nextChapterTime)
                        {
while(lapsedTime.GetTotalSeconds() <= nextChapterTime)
                                {
checkCancel();
                                        Sleep(500);
                                        time =
CTime::GetCurrentTime();
                                        lapsedTime =
time - startEvent;
pLayer->updateTime(lapsedTime.GetTotalSeconds(),title);
pLayer->put_serverTime(time.GetTime(),title, chapter ,lapsedTime.GetTotalSeconds() *
1000,  eventLength);
                                }
pLayer->sendCommand(command);
                                                                Sleep(500);
                                                        }
                                                        time =
CTime::GetCurrentTime() ;
                                                        ::SysFreeString(command);
                                                }
                                                else
                                                {
                                                        TRACE("no more entries in
the chapter table");
                                                        break;
                                                }
                                        }
                                }

                                ::OutputDebugString("End Session");
                                pLayer->endSession(NULL);
                        }
                        catch(ExceptionCanceled * pExcp)
                        {
                                delete pExcp;
                                _bstr_t msg = "Cancellation occured";
                                pLayer->endSession(msg);
                        }
                        catch(...)
                        {
                                _bstr_t msg = "Cancellation occured";
                                pLayer->endSession(msg);
                        }
                }

        }

};
/////////////////////////////////////////////////////////////////////////////
// CBusinessLayer

STDMETHODIMP CBusinessLayer::InterfaceSupportsErrorInfo(REFIID riid)
{
        static const IID* arr[] =
        {
                &IID_IBusinessLayer
        };
        for (int i=0; i < sizeof(arr) / sizeof(arr[0]); i++)
        {
                if (InlineIsEqualGUID(*arr[i],riid))
                        return S_OK;
        }
        return S_FALSE;
}

HRESULT CBusinessLayer::FinalConstruct()
{

        HRESULT hr = CoCreateInstance(CLSID_CConfigMgrImpl,
                                                                0,
                                                                CLSCTX_ALL,
IID_ICConfigMgrImpl,
(void**)&m_pICConfigMgrImpl);

        return hr;
}

void CBusinessLayer::FinalRelease()
{
        if (m_pthread)
                m_pthread->stop();
        delete m_pthread;
        if (m_pICConfigMgrImpl)
                m_pICConfigMgrImpl->Release();
        try
        {
                if (m_pIDBConnect)
                        m_pIDBConnect->Release();
        }
        catch(...)
        {
        }
}

void CBusinessLayer::ChkValidEvent()
{
        ::OutputDebugString("\n Check valid Event\n");
        m_firstTime = false;
        if (m_diskID.length() == 0)

                throw new IAUserException("Invalid Disk id");

        if (m_pICConfigMgrImpl)
        {
                std::string debugmsg;
                debugmsg = "disk id=" + m_diskID + "; location id = " + m_locationID
+  "\n";
                ::OutputDebugString(debugmsg.c_str());
                m_pICConfigMgrImpl->put_diskID(m_diskID);                //
Variable used for search critera
                m_pICConfigMgrImpl->put_locationID(m_locationID); // Variable used
for search critera
                m_pICConfigMgrImpl->get_hostType(&m_hostType);

                if (m_hostType)
                {
                        //
                        // Create a DBConnector, store the pointer for future use.
                        // Store values from db.
                        //
                        ::OutputDebugString("\n Host type is checked\n");
                        HRESULT hr = S_OK;
                        if (!m_pIDBConnect)
                        {
                                hr = CoCreateInstance(CLSID_DB_Connector,
                                                                            0,
                                                                            CLSCTX_ALL,

IID_IDB_Connector,

(void**)&m_pIDBConnect);
                        }
                        if (SUCCEEDED(hr))
                        {
```

```
                                        BusinessLayer
                                    ::OutputDebugString("\n Initialize DB Connector\n");
                                    m_pIDBConnect->put_diskID(m_diskID);            //
    Variable used for search critera
                                    m_pIDBConnect->put_locationID(m_locationID); //
    Variable used for search critera
                                    m_pIDBConnect->chkEvent();
                                    BSTR data;
                                    m_pIDBConnect->get_diskID(&data);
                                    if (data)
                                    {
                                            m_diskID = data;
                                            ::SysFreeString(data);
                                    }
                                    m_pIDBConnect->get_locationID(&data);
                                    if (data)
                                    {
                                            m_locationID = data;
                                            ::SysFreeString(data);
                                    }
                                    long time;
                                    m_pIDBConnect->get_startEvent(&time);
                                    m_startEvent = time;
                                    m_pIDBConnect->get_stopEvent(&time);
                                    m_stopEvent = time;
                                    m_pIDBConnect->get_thresold(&threshold);
                                    m_pIDBConnect->get_hostType(&m_hostType);

                                    long * nDecoderArray;
                                    long * nCapabilitiesArray ;
                                    nDecoderArray = nCapabilitiesArray = NULL;

                                    if
    (SUCCEEDED(m_pIDBConnect->decoderArray(&nDecoderArray, &nCapabilitiesArray)))
                                    {
                                            int i = 0;
                                            while(nDecoderArray[i] != -1)
                                            {
                                                    m_capabilities[nDecoderArray[i]] =
    nCapabilitiesArray[i];
                                                    i++;
                                            }
                                            CoTaskMemFree(nDecoderArray);
                                            CoTaskMemFree(nCapabilitiesArray);
                                    }

                                    ::OutputDebugString("\n Prepare to start thread\n");

                                    m_pthread = new layerthread;
                                    m_pthread->start(this,false);
                            }
                            else
                            {
                                    throw new COMException(hr);
                            }

                    }
                    else
                    {
                            //
                            // Create a Reference Connector, and store the pointer for
    future use.
                            // TBD
                            //
```

```
                }
        }
}

STDMETHODIMP CBusinessLayer::get_disk(BSTR* pVal)
{
        // TODO: Add your implementation code here
        *pVal = m_diskID.copy( );
        return S_OK;
}

STDMETHODIMP CBusinessLayer::put_disk(BSTR newVal)
{
        // TODO: Add your implementation code here
        m_diskID = newVal;
        return S_OK;
}

STDMETHODIMP CBusinessLayer::get_location(BSTR* pVal)
{
        // TODO: Add your implementation code here
        *pVal = m_diskID.copy( );
        return S_OK;
}

STDMETHODIMP CBusinessLayer::put_location(BSTR newVal)
{
        // TODO: Add your implementation code here
        m_locationID = newVal;

        return S_OK;
}

STDMETHODIMP CBusinessLayer::get_startEvent(long *pVal)
{
        // TODO: Add your implementation code here
        *pVal = m_startEvent.GetTime(); // m_pIDBConnect->get_startEvent(pVal);
        return S_OK;
}

STDMETHODIMP CBusinessLayer::put_startEvent(long newVal)
{
        // TODO: Add your implementation code here
        time_t time = newVal;
        m_startEvent = time;
        return S_OK;
}

STDMETHODIMP CBusinessLayer::get_stopEvent(long *pVal)
{
        // TODO: Add your implementation code here
        *pVal = m_stopEvent.GetTime(); // m_pIDBConnect->get_startEvent(pVal);
        return S_OK;
}

STDMETHODIMP CBusinessLayer::put_stopEvent(long newVal)
{
        // TODO: Add your implementation code here
        time_t time = newVal;
        m_stopEvent = time;
        return S_OK;
}
```

```
STDMETHODIMP CBusinessLayer::get_threshold(long *pVal)
{
        // TODO: Add your implementation code here
        *pVal = threshold; // m_pIDBConnect->get_thresold(pVal);
        return S_OK;
}

STDMETHODIMP CBusinessLayer::put_threshold(long newVal)
{
        // TODO: Add your implementation code here
        threshold = newVal;
        return S_OK;
}

HRESULT CBusinessLayer::GetNextPair(long *theTime, long *nTitle, long * nChapter,
BSTR *chapterCmnd)
{
        return m_pIDBConnect->get_NextChapter(theTime,nTitle,nChapter,chapterCmnd);
}



_bstr_t CBusinessLayer::firstdvdCmd()
{
        //
        // Execute the first DVD Command
        //
        BSTR msg = NULL;
        _bstr_t dvdMsg;
        m_pIDBConnect->get_initialDVDCommand(&msg);
        if (msg)
                dvdMsg = msg;
        return dvdMsg;
}

void CBusinessLayer::sendCommand(BSTR szMsg)
{
        Fire_sendCommand(szMsg);

}

void CBusinessLayer::endSession(BSTR  szMsg)
{
        Fire_endSession(szMsg);

}


void CBusinessLayer::updateTime(LONG  time, long nTitle)
{
        Fire_updatetime(time,nTitle);
}

STDMETHODIMP CBusinessLayer::Initialize()
{
        HRESULT hr = S_OK;
        try
        {
                ChkValidEvent();
        }
        catch(IAUserException *pexcpt)
        {
                delete pexcpt;
                hr = E_FAIL;
                _bstr_t msg = "USER exception occured\n";
                Fire_endSession(msg);
        }
        catch(COMException * pcomexcpt)
        {
                hr = pcomexcpt->operator HRESULT();
                _bstr_t msg = "COM exception occured\n";
                Fire_endSession(msg);
        }
        catch(...)
        {
                _bstr_t msg = "Unknown exception occured\n";
                Fire_endSession(msg);

        }

        return hr;
}

STDMETHODIMP CBusinessLayer::TranslateTimePlay(long nDecoderType, long nTitle, long
nTime, BSTR *szCmd)
{
        // TODO: Add your implementation code here
        HRESULT hr = E_FAIL;
        decoderCapabilities::iterator it = m_capabilities.find(nDecoderType);
        if (it != m_capabilities.end())
        {
                if ((*it).second == 0)
                {
                        char translate[_MAX_PATH];
                        sprintf(translate, "tmp:%d:%d", nTitle, nTime * 1000);
                        *szCmd = _bstr_t(translate).copy();
                        hr = S_OK;
                }

        }

        return hr;
}

STDMETHODIMP CBusinessLayer::get_eventLength(long *pVal)
{
        // TODO: Add your implementation code here
        m_timeLock.lock();
        *pVal = m_eventLength.GetTime();
        m_timeLock.unlock();
        return S_OK;
}

STDMETHODIMP CBusinessLayer::get_lapsedTime(long *pVal)
{
        // TODO: Add your implementation code here
        m_timeLock.lock();
        *pVal = m_lapsedTime.GetTime();
        m_timeLock.unlock();

        return S_OK;
}
```

BusinessLayer

```
STDMETHODIMP CBusinessLayer::get_chapterProperties(long *pVal)
{
	// TODO: Add your implementation code here
	m_timeLock.lock();
	*pVal = m_chapter;
	m_timeLock.unlock();
	return S_OK;
}

STDMETHODIMP CBusinessLayer::get_titleProperties(long *pVal)
{
	// TODO: Add your implementation code here
	m_timeLock.lock();
	*pVal = m_title;
	m_timeLock.unlock();
	return S_OK;
}

STDMETHODIMP CBusinessLayer::get_serverTime(long *pval)
{
	// TODO: Add your implementation code here
	m_timeLock.lock();
	*pVal = m_serverTime.GetTime();
	if (*pVal == 0)
	{
		char msg[1024];

		sprintf(msg, "title = %d,  chapter = %d, location = %s\n",m_title,
m_chapter, m_locationID.operator char *());
	}
	m_timeLock.unlock();

	return S_OK;
}
void CBusinessLayer::put_serverTime(/*[in]*/ long serverTime, long title, long
chapter, long lapsedTime, long length)
{
	m_timeLock.lock();
	m_serverTime = serverTime;
	m_title = title;
	m_chapter = chapter;
	m_lapsedTime = lapsedTime;
	m_eventLength = length;
	m_timeLock.unlock();
}
```